Java Script

JavaScript (often abbreviated as JS) is a high-level, versatile, and widely-used programming language primarily known for its role in web development. It enables interactive and dynamic behavior on websites.

In JavaScript, `var` is function-scoped and was traditionally used to declare variables. `let` and `const` are block-scoped. The key difference between `let` and `const` is that `let` allows for reassignment while `const` creates a read-only reference.

JavaScript is a programming language initially designed to interact with elements of web pages. In web browsers,

- **The Document Object Model (DOM) provides interfaces for interacting with elements on web pages**
- **The Browser Object Model (BOM) provides the browser API for interacting with the web browser.**

Function

functions are blocks of reusable code that perform a specific task. Functions can take parameters (inputs), execute a series of statements, and return a value.

```
function Wish(){
    console.log("hello.....");
}

Wish()
```

```
// Function Declaration
function greet(name) {
  console.log("Hello, " + name + "!");
}

// Function Call
greet("John"); // Output: Hello, John!
```

```
// Function Expression
const add = function (a, b) {
    return a + b;
  };

  // Function Call
  const result = add(3, 5);
  console.log(result); // Output: 8
```

Arrow Function

```
// Arrow Function
const multiply = (x, y) => x * y;

// Function Call
const product = multiply(4, 6);
console.log(product); // Output: 24
// Function with Rest Parameters
function sum(...numbers) {
   return numbers.reduce((total, num) => total + num, 0);
 }

 // Function Call
 const total = sum(1, 2, 3, 4, 5);
 console.log(total); // Output: 15
```

if statement
if...else statement
if...else if... statement.

If Statement

```
let temperature = 35;

if (temperature > 30) {

  console.log("It's a hot day!");

}
```

Else if

```
let temperature = 25;

if (temperature > 30) {

console.log("It's a hot day!");

} else {

console.log("It's not too hot today.");

}
```

If elseif

```
  let temperature = 25;

if (temperature > 30) {

  console.log("It's a hot day!");
```

```
} else if (temperature <= 30 && temperature >= 20) {
  console.log("The weather is pleasant.");
} else {
  console.log("It's cold today.");
}
```

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 20;

            if( age > 18 ) {
               document.write("<b>Qualifies for
driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and
then try...</p>
   </body>
</html>
```

JavaScript Switch

the switch statement is used for conditional branching. It provides a way to execute different code blocks based on the value of an expression.

```
let day = 3;

let dayName;

switch (day) {

 case 1:

   dayName = 'Monday';

   break;

 case 2:

   dayName = 'Tuesday';

   break;

 case 3:

   dayName = 'Wednesday';

   break;

 case 4:

   dayName = 'Thursday';

   break;
```

```
  case 5:
    dayName = 'Friday';
    break;
  default:
    dayName = 'Weekend';
}
console.log(dayName); // Output: Wednesday
```

JS Array Methods

```
// let fruits = ['apple', 'banana', 'cherry','mango'];
// console.log(fruits[0]);
// console.log(fruits[2]);



// fruits.push('date');
// console.log(fruits);



// let lastFruit = fruits.pop();
// console.log(lastFruit);
// console.log(fruits);
// console.log(fruits)
// let firstFruit = fruits.shift();
// console.log(firstFruit);
// console.log(fruits);
```

```
// fruits.unshift('apple');
// console.log(fruits);

// let position = fruits.indexOf('cherry');
// console.log(position);




// fruits.splice(1,2)
// console.log(fruits)



// fruits.splice(0,1, 'blueberry', 'blackberry');
// console.log(fruits);

// let newFruits = fruits.slice(0, 2);
// console.log(newFruits);



// fruits.forEach(function(fruit) {
// console.log(fruit);
// });

// let upperCaseFruits = fruits.map(function(fruit) {
// return fruit.toUpperCase();
// });
// console.log(upperCaseFruits);
```

```
// let numbers = [1, 2, 3, 4, 5];
// let sum = numbers.reduce(function(total, num) {
// return total + num;
// }, 0);

// console.log(numbers)
// console.log(sum);



// let foundFruit = fruits.find(function(fruit) {
// return fruit.startsWith('');
// });
// console.log(foundFruit);



// let fruits = ['apple', 'banana', 'cherry', 'blueberry', 'blackberry'];



// let filteredFruits = fruits.filter(function(fruit) {
// return fruit.startsWith('b');
// });

// console.log(filteredFruits);

// let nestedArray = [1, [2, [3, [4, 5]]]];
```

```
// let flatArray = nestedArray.flat(Infinity);
// console.log(flatArray);
```

JavaScript | Callbacks

JavaScript is an asynchronous language, which means that it can handle multiple tasks simultaneously. Callbacks are a fundamental aspect of JavaScript, as they allow you to run code after an asynchronous operation has been completed. In this article, we'll look at what callbacks are, why they're used, and how to use them with real-life examples and code examples.

call back

normal two function

```
function a(){
console.log("inside function A");
}

function b(){
console.log("inside function B");
}

a()
b()
```

CallBack Function

```
function a(callBack){
callBack()
console.log("inside function A");

}

function b(){
console.log("inside function B");
}

a(b)
```

we can pass anonymous function as callback

```
function a(cb){
cb()
console.log("inside Function A")

}

a(function(){
console.log("test Callback");
})
```

```
const posts=[
{id:1,title:"indroduction"},
{id:2,title:"chapter-1"}
]

const getPost=()=>{
setTimeout(()=>{
console.log(posts)
},1000)
}
// getPost()

const addPost =(post)=>{
```

```
setTimeout(()=>{
posts.push(post)
},2000)
}

addPost({id:3,title:"chapter-2"})

getPost()


call back


const posts=[
{id:1,title:"indroduction"},
{id:2,title:"chapter-1"}
]

const getPost=()=>{
setTimeout(()=>{
console.log(posts)
},1000)
}
// getPost()

const addPost =(post,callBack)=>{
setTimeout(()=>{
posts.push(post)
callBack()
},2000)
}

addPost({id:3,title:"chapter-2"},getPost)

const posts=[
{id:1,title:"indroduction"},
{id:2,title:"chapter-1"}
]

const getPost=()=>{

console.log(posts)

}
```

```
const addPost =(post)=>{

posts.push(post)


}

addPost({id:3,title:"chapter-2"})
getPost()




console.log("Start");

// Simulating a blocking operation

for (let i = 0; i < 100; i++) {
console.log(i)
}

console.log("End");



console.log("Start");

setTimeout(() => {
for (let i = 0; i < 100; i++) {
console.log(i)
}

console.log("Async operation completed");
}, 2000);

console.log("End");

// Function to be executed
function sayHello() {
console.log("Hello, world!");
}
```

```
// Call the function every 2 seconds (2000 milliseconds)
let intervalId = setInterval(sayHello, 1000);

// To stop the interval after 10 seconds (10000 milliseconds)
setTimeout(() => {
clearInterval(intervalId);
console.log("Interval cleared.");
}, 10000);




const myPromise = new Promise((res, reject) => {
// Simulate an asynchronous operation
setTimeout(() => {
const success = true; // Change to false to simulate failure
if (success) {
res("Operation succeeded!");
} else {
reject("Operation failed.");
}
}, 2000);
});


myPromise
.then((result) => {
console.log(result); // "Operation succeeded!"
})
.catch((error) => {
console.error(error); // "Operation failed."
})
.finally(() => {
console.log("Operation completed."); // Runs regardless of success or failure
});



const promise1 = Promise.resolve("Promise 1");
const promise2 = Promise.resolve("Promise 2");
const promise3 = Promise.resolve("Promise 3");

Promise.all([promise1, promise2, promise3])
```

```
.then((results) => {
console.log(results); // ["Promise 1", "Promise 2", "Promise 3"]
})
.catch((error) => {
console.error(error);
});


const promise1 = new Promise((resolve) => setTimeout(resolve, 100, "First"));
const promise2 = new Promise((resolve) => setTimeout(resolve, 200, "Second"));

Promise.race([promise1, promise2])
.then((result) => {
console.log(result);
});




const promise1 = Promise.resolve("Resolved");
const promise2 = Promise.reject("Rejected");

Promise.allSettled([promise1, promise2])
.then((results) => {
console.log(results);

});




const promise1 = Promise.reject("Error 1");
const promise2 = Promise.resolve("Success");
const promise3 = Promise.reject("Error 2");


Promise.any([promise1, promise2, promise3])
.then((result) => {
console.log(result); //
})
.catch((error) => {
```

```
console.error(error); // AggregateError if all promises are rejected
});




// const promise1= new Promise((res,rej)=>{
// res()
// })

// promise1.then(()=>{
// console.log("promise Success");
// }).catch(()=>{
// console.log("Promise Rejected");
// }).finally(()=>{
// console.log("promise work");
// })

// after execution if the promise is resolve .then will work
// after execution if the promise is rejected .catch will work




const posts = [
{ id: 1, title: "indroduction" },
{ id: 2, title: "chapter-1" },
];

const getPost = () => {
setTimeout(() => {
console.log(posts);
}, 1000);
};

// getPost()

const addPost = (post) => {
return new Promise((resolve, reject) => {
```

```
setTimeout(() => {
posts.push(post);

const err = false;
if (err) {
reject();
} else {
resolve();
}
}, 2000);
});
};

addPost({ id: 3, title: "chapter-3" })
.then(getPost)
.catch(()=>{
console.log("error");
}).finally(()=>{
console.log("Promise Working");
})
```